Tecnológico de Monterrey

# Act 6.2 - Reflexión Final de Actividades Integradoras de la Unidad de Formación TC 1031

Guillermo Esquivel Ortiz | A01625621

Programación de estructuras de datos y algoritmos fundamentales

Gpo 570

Eduardo Arturo Rodríguez Tello

Para la afrontar problemas informáticos hay una multitud de soluciones a tomar, existen múltiples posibilidades a seleccionar entre las cuales cada una de estas soluciones portará sus determinadas complejidades computacionales, por lo cual, no todas seran las mas apropiadas a tomar a causa del tiempo, energía o del poder computacional que se porte. Para tomar en cuenta de mejor manera esta, se pueden analizar las complejidades computacionales de los algoritmos que lleguen a solucionar el problema en cuestión, además del uso de distintas estructuras de datos para adaptarse mucho mejor al problema.

El problema que nos aconteció para esta materia fue el estar presentes en una vulnerabilidad al sistema que estamos trabajando (me sonaba mucho a un DDos) en el cual se debía analizar los intentos de conexion de ciertas direcciones ip y en base a ello, determinar múltiples factores, como el boost master o la distancia que se tiene ante el resto de ip's.

Primero que nada hay que contemplar que hoy en día, al situarnos en un mundo tan interconectado, existen enormes cantidades de información que, de forma manual, resultan prácticamente imposibles de ordenar o buscar los elementos deseados y en un ambiente de seguridad informática, los algoritmos de ordenamiento y búsqueda son una herramienta fundamental para ayudar a identificar y clasificar información de forma eficiente, lo que nos permite detectar amenazas y evitar ataques a los sistemas informáticos, estos algoritmos también ayudan a reducir los tiempos de procesamiento, lo que los hace ideales para analizar grandes volúmenes de datos y detectar patrones de comportamiento anómalos. Por ejemplo, en el análisis de la bitácora, resultó ser óptimo usar el Merge Algorithm para el ordenamiento de la información, debido a su complejidad computacional O(n log n, seguido de un algoritmo de búsqueda como la búsqueda binaria, cuyo tiempo de procesamiento es O(log n) resultando ser los más eficientes en primer instancia.

Para desarrollar mejores soluciones ante estos problemas se puede tomar el uso de múltiples data structures como lo comentamos anteriormente, tenemos ejemplos como las listas ligadas, arbolés binarios, stocks, etc. En este caso, tomamos una doble lista ligada para guardar los objetos y procesarla mediante dos algoritmos de ordenamiento, mergeSort y quickSort; en un comienzo se pensaba que se tendría resultados favorables por ambos casos al tratarse de algoritmos con una complejidad computacional O(nlogn) como se había comentado anteriormente, pero específicamente para esta situacion, hubo una mejor respuesta por parte de QuickSort teniendo una notable mejoría en un gran cúmulo de información. Respecto a nuestra búsqueda de datos, se procuró realizar con el algoritmo BinarySearch.

Para optar otra mejora a nuestra solucion, se opto dar el uso a otra estructura de datos, en este caso un arbol binario. Según Osvaldo Cairo y Silvia Guardati un árbol de estructura de datos se puede definir como una estructura jerárquica y de forma no lineal, que se aplica en uno o varios elementos u objetos que se llaman nodos. Estos se consideran no lineales y dinámicos de datos muy importantes, estos suelen ser muy utilizados en informática para poder hacer búsquedas grandes y complejas, de hecho, casi todos los sistemas operativos almacenan sus archivos en árboles o estructuras que son muy parecidas a estas. Como se mencionó anteriormente, se les llama dinámicas porque si metemos un nuevo dato estas pueden cambiar de forma, tamaño, orden, durante la ejecución del programa. Y se les llama no lineales porque cualquier elemento del árbol puede tener más de un sucesor, a diferencias de las linked list, queue, stacks, etc. que almacenan de manera lineal. Estos se caracterizan por tener nodos, que pueden ser nodos padres, hijos o hermanos, y estos pudiendo ser raíz, rama u hoja. Los podemos dividir en niveles que nos darán la altura del árbol. Hay dos tipos de árboles que son los binarios y los multicaminos con sus variaciones cada uno. Estos son muy importantes ya que se pueden utilizar en los sistemas de archivos de los sistemas operativos, que están compuestos por jerarquías y archivos, también en la jerarquía de clases en los lenguajes orientados a objetos, etc. También al ser de las más utilizadas, eleva su grado de importancia en la informática, aunque pueden llegar a ser complejas. Los ataques cibernéticos son más comunes año con año desgraciadamente, que incluso nos puede pasar a nosotros mismos. Para poder detectar si nuestra red esta infectada podríamos checar ciertos nodos que nos resalten información, por ejemplo, si vemos que una IP se repite demasiado, puede que se trate sobre un ataque consistente, pudiendo banear la IP para proteger nuestra red. Si analizamos los errores de esa IP podemos ver que esta tratando de hacer exactamente para poder neutralizarlo lo antes posible. Ahora, determinada nuestra situacion al estar en un caso de seguridad, podemos llegar a aplicar un registro de todas las entradas tomando en cuenta sus Ip’s, obtener las que tengan mayor numero de registros y asi poder bloquearlas para evitar alguna vulnerabilidad a nuestro sistema .

Pasando por el uso de tales estructuras de datos, podemos proceder a dar uso los grafos, los cuales son una estructura de datos que toma multiples conexiones entre distintas variables, creando una tipo de red en la cual puedes moverte por distintos nodos. Tomando en cuenta la naturaleza de esta estructura de datos, la denotamos perfecta para el problema que nos acontece, una serie de ip’s tomando direccion a otra serie de ip’s con su determinada distancia (posiblemente tiempo), esto seria un caso complicado de analizar para otro tipo de estructura de datos. Nuestro grafo lo desarrollamos a partir de una lista de adyacencia la cual trabaja en base de una LinkedList y un

template pair, los cuales nos apoyaran para el almacenamiento de todas las ip's en sus respectivos nodos y vertices. Este tipo de grafo se elogio gracias a que es mas eficaz que una matriz de adyacencia y es mas simple de implementar que una lista de arcos. Para primeramente tomar registro de las ip's con mayor numero de salidas, recurrimos a tomar el apoyo de un Arbol Heap, el cual nos ayudara a sortear los elementos en una complejidad de O(nlogn) para finalmente obtener el numero de grados que tiene cada Ip. Teniendo tal informacion, podemos recurrir a encontrar el Boost Master (el que tomamos en cuenta como la ip con mayor riesgo) y con el algoritmo Dijkstra, revisar cuales son las distancias que tiene ante el resto de ip's para asi poder considerar cuales ip's son las mas seguras o cuales contienen un mayor riesgo de ataque.

En base a nuestro codigo anterior en el cual dimos uso a la estructura de datos de grafos, podemos proceder a dar uso del Hash Table la cual es finalmente la ultima que se probara en este problema. El hash table tiene la posibilidad de asociar keys con sus respectivos valores, se trata de series de conjuntos las cuales cuentas con sus respectivas relaciones. La ventaja que se tiene en el uso de este tipo de estructura de datos son las busquedas, ya que se para buscar un dato solo se necesitaria tener como intermediario una funcion hash la cual permitira obtener directamente el key y de tal manera realizar la busqueda sin recorrer multitud de elementos. Ante esta situacion, solo acontece un problema el cual son las colisiones, estas suceden cuando las keys resultan ser las mismas luego de aplicar la funcion hash y se obtiene los datos incorrectos, pero para esto existen multiples tecnicas o buenas practicas para disminuir la posibilidad de colision.

Ahora situandonos en nuestro problema, habiamos podido obtener multiples datos de importancia usando un grafo, ahora en este caso se trabajara en base al uso de un grafo, guardando los datos en tal estructura de datos para luego pasarlas a un hash table y resumir la informacion de relevancia obtenida a traves de la hashtable.

Finalmente, dentro de todo este conjunto de actividades, se pudo dar un exhaustivo analisis a las multiples estructuras de datos y algoritmos que dimos uso, para el problema que nos acontecio considero que el uso de grafos fue la estructura mas optima para guardar los datos con los que trabajamos, para mas tarde procesar la informacion mediante algoritmos como dijkstra algorithm o mas tarde en una hash table que si se soluciona el problema de las colisiones, resulta perfecto para nuestro caso de trabajo.

Las mejoras que se podrian implementar probablemente podria ser en cambiar la funcion hash para probar cual puede resultar mas eficiente al contener un menor numero de colisiones o inclusive cambiar nuestro grafo de una lista de adyacencias a una edge list para tener un procedimiento aun mas eficiente.

**References**

*Hernández, D. V. L. C. M., Guerra, G. L. H., & Gurrión, S. E. G. (2020). Estructuras de datos y algoritmos fundamentales. Editorial Digital del Tecnológico de Monterrey.*

*Why do we need searching algorithms? - Searching - KS3 Computer Science Revision. (n.d.). BBC. Retrieved January 18, 2023, from https://www.bbc.co.uk/bitesize/guides/zgr2mp3/revision/1*

*Why do we need sorting algorithms? - Sorting - KS3 Computer Science Revision. (n.d.). BBC. Retrieved January 18, 2023, from https://www.bbc.co.uk/bitesize/guides/z2m3b9q/revision/1*